|  |
| --- |
| School of Software Design and Data Science |
| **Full stack JS App** |
| WEB422 - Project |
|  |
| **Daniel Park, Angela Marie Tangudin Chua, Amany Omar**  **Submission Date** |
| **4/15/2025** |

|  |
| --- |
| This document explains how to build a frontEnd dashboard that consume backend API …………………………. |

Table of Contents

[Task 1: Pick Your Store and Product/Service Inventory 2](#_Toc174091863)

[Task 2: Project Planning, Wireframing, Layout, Dynamic Content 3](#_Toc174091864)

[Task 3: Authentication/Authorization 4](#_Toc174091865)

[Task 4: "Favourites" functionality 5](#_Toc174091866)

[Task 5: "Route Guard" functionality 6](#_Toc174091867)

[Task 6: deployment 7](#_Toc174091868)

[Bonus questions 8](#_Toc174091869)

[Project planning/sharing tasks 9](#_Toc174091870)

[Project performance check 10](#_Toc174091871)

[Summary 11](#_Toc174091872)

# Task 1: Pick Your Store and Product/Service Inventory

(Describe the major steps for identifying product/service/API)

For this project, we chose a marketplace model that offers a variety of Pokemon cards. The major steps for identifying our services, products, and API were:

1. **Initiation:** We brainstormed multiple different ideas and finalized on creating a marketplace for Pokemon cards as it was a shared interest and we felt that it was a niche product to work with.
2. **Research:** We discussed and researched through a variety of API’s and products that we could use for our Pokemon marketplace but ended up finalizing and choosing the pokemon TCG API. This API provided all of the information we needed along with data sets of the products (cards) we were going to list.
3. **Service Inventory:** We defined our product inventory based on the available data and information from the API, making sure that our inventory covers necessary details such as name, image, description, prices, and categories for each card.

# Task 2: Project Planning, Wireframing, Layout, Dynamic Content

(Describe the project planning (Gantt Chart) and major steps for designing the layout (Figma) and visualizing data using Card design)

1. **Project Planning:** We made a Jira Kanban board to keep track of tasks to do and completed tasks, assigning tasks to specific members of our team.
2. **Wireframe:** Using Figma we created wireframes for our website to create a basis of how our UI should look.
3. **Dynamic Content:** 
   1. Homepage: We displayed four random cards using math.random to generate random cards for the featured cards section, the cardlist component comes in handy here as we can re-use it and pass in the new array that we built that is sliced of at 0, 4. To prevent hydration errors that I created a cardclient page which pretty much just tells it to only load it client side.
   2. Card Page: We display all of our cards with a sidebar that can filter by card category and max price, and a search bar with history for logged in users. We also have pagination to chunk the large dataset and make it more readable by having only 30 per page. The filters and search update the dataset and then we again apply pagination to the results.
   3. Search and Filter Logic: After apply the search and filter logic the result is passed to pagination.
   4. Individual card details page: Each card has its own unique page at card/[id] rendered using dynamic routing. We show the image and full details of the card and buttons to interact with it. I wanted to add a price history section as Pokemon cards are similar to stocks or various marketplaces where the price rises and falls according to the previous purchases. However, this was not possible as the API’s that had the previous sale history were not open, and the only other way to implement this would be to keep track of sales on our webpage (which we have not implemented).
   5. Pagination: We manually implement the pagination here with a constant cardsPerPage = 30, and the current page is tracked via useState. We then have slice logic to display only cards on our current page. The previous and next buttons are also disabled accordingly.

# Task 3: Authentication/Authorization

(Describe the major steps on how to make JWT user authentication in the app)

**User Registration:** In register.js and user.js a new user submits their username and password into the form, meanwhile in the back end we hash the password using bcryptjs and then we save it to our MongoDB database. In the front end register.js will send the registration request with registerUser() in authenticate.js.

**User Login and JWT Token:** The front end will send a POST request with the username and password to our api/user/login using the authenticateUser() function.

If this is valid the backend make a JWT token, and the token is returned and stored in local storage.

**Session State and Atoms:** Once a user is logged in successfully the app sets the atoms states loggedInAtom, favouritesAtom, searchHistoryAtom by using getFavourites() and getHistory().

**Authorization:** For our protected routes like getting the users favourites and search history the client sends the token in the request header.

**Logout:** The token is removed from local storage.

# Task 4: "Favourites" functionality

(Describe the major steps on how to make “favorite products” in the app)

**Favourites Data Model:** In user.js each user document has favorites: [{ cardId: { type: String } }], which will hold the list of the favourite cards by ID per user in MongoDB

**FrontEnd:** In favoritesAtom.js we use jotai to track favourite cards client side, export const favoritesAtom = atom([]); This global state allows components updates and reads favourite cards.

**Displaying Favourites:** If the user is not logged in they will be redirected to the login page, if they are logged we call /api/user/get/favorites with the JWT token, for every cardID that is returned we call {cardId} to get the full card data, and display results using CardList.

**Toggle Favourites:** If a user is not logged in the favourites toggle button will be disabled, however when a logged in user clicks the toggle button we get the token from local storage, if the card is already in the favourites list it sends a delete request, if not it sends a put request.

**Task 5: "Route Guard" functionality**

(Describe how to implement protected routes)

To ensure only registered users could access features like searching and favoriting Pokemon cards, we implemented a Route Guard in our app using Next.js and Jotai to restrict the favorite page and search page to authenticated users only, by redirecting the user to login page when attempting to search or favorite.  
  
Steps:

1. Created RouteGuard.js Component inside components/ folder. It wraps around the main layout and checks user login status using loggedInAtom from jotai.
2. Used PUBLIC\_PATHS array including to only protect the /favorites and /search pages. This made the homepage and public content accessible without foricing the user to log in immediately.
3. Used useEffect to check login status
4. Reused the updateAtoms() function from our login page to re-sync user data (favorites and history) to ensure favorites didn’t disappear after refreshing the page.

# Task 6: deployment

(Describe the major steps for deployment)

Once the Poke Mart project was initialized, added and pushed to github repository. Created initial homepage with “Website coming soon!” and deployed to Vercel.

Steps:

1. Created initial homepage with initial message.
2. Pushed our code to GitHub under a private repository
3. Signed into vercel.com using GitHub authentication
4. Imported our GitHub repo to Vercel and selected the correct project directory to deploy.
5. Clicked “Deploy”
6. Vercel generated live URL that automatically updates evertime we push to main.

\*Learned how to read, understand, and debug deployment errors

# Bonus questions

(Describe the major steps for designing the bonus question)

The goal of the bonus question was to add dynamic client side filtering to our card list so that the users can filter cards by category or limit results by maximum price. This will display results based on filter without having to refresh the page.

First I created a re-useable sidebar component that tracks the selected filters in a local state, every time the user ticks a checkbox or moves the slider the component will call onFilterChange, this is a callback passed from the parent to update the filtered view. In /pages/cards/index.js, we apply filters when filters change using a useEffect, this makes sure that the visible card list always reflects the current state of the filter. After filtering we slice the new filtered array to paginate it, then that is passed to our existing cardList to neatly display our filtered array.

# Project planning/sharing tasks

(Describe how did you divided the work and did project planning. Have you changed the day-1-planning/milstones/deliverables that we did at the beginning of project in the class?)

We did not deviate too much from the original tasks. We delivered milestone deliverables on time. My original tasks were to implement routeguards but instead we decided that I would implement most of the design for the pages and components and Amany took over that role instead.

# Project performance check

(check your project performance using google lighthouse and add the screenshot of your current app status here )

A screenshot of a website

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# Summary

(Describe your experience in this project, share your feedback on how this project was align with the course objective and how do you think the project can be changed to provide a better experience, …)

Daniel- This was a really great experience for me as it helped me learn more about working together as a team on a full stack web application. I would have liked to work more with the backend side of the project but I also enjoyed implementing the front end and making things look nice.

During our recording we forgot to evaluate ourselves, we would rate our project a

8/10 as we accomplished all of the required functionality with a professional looking front end.